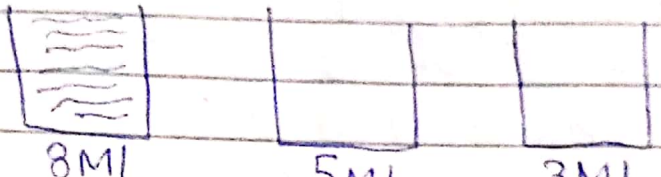
OOP (Object Oriented Programming)

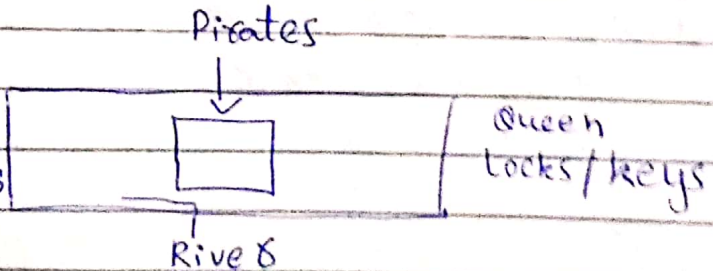
Syeda Tayyaba Bukhari
tayyaba.bukhari@nu.edu.pk

Lecture 1



8ML water	No water	No water
8	0	0
3	5	0
3	2	3
6	2	0
6	0	2
1	5	2
1	4	3
4	4	0

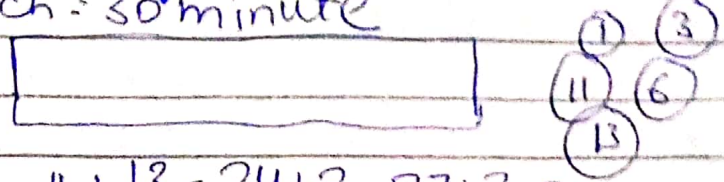
King/Queen Problem:



∵ king wants to give a ring to queen

Bull Crossing Problem

Torch - 30 minute



3+1+6+1 = 11+13 = 24+3 = 27+3 = 30

Declaration Initialization

```
int a= 10;
// Display content of a
cout<<a; // 10
// Display size of a
cout<<sizeof(a); // 4
// Display address of a
cout<<&a;
```

built in function

Content

10

a -> name

// 4 -> It means 4 bytes. Always show result in bytes.

& = ampersand

Address Operator

Note:

8 bits = 1 byte

4 bits = 1 Nibble

Integer -> format -> whole

& (address operator) -> format -> Hexadecimal [0-9, A-F]

```
int a =10;
  int b=a;
cout<<b; // 10
```

10	10
a	b

Restriction: a ki value print krwani bt ksi dusre variable ke through.

Restriction: Pehle address ko dusre variable me save krwana phir print krwana.

For this purpose we need a variable with datatype hexadecimal ~~as datatype of & is hexadecimal~~ but there is no such datatype so we use specific variable to solve this issue called pointers.

Pointer Variable:

A special variable that deals with addresses only.

Syntax:

datatype of variable who's address we want to store * name of variable;

(* : asteric)

variable of integer type

int a = 10;

pointer to integer

int *p; =>

p = &a;

cout << p; // address of variable

Two methods to write:
int *p; or int * p
(space before *) (space after *)

Note:

int * p } wrong / Error
(space before *, space after *)

Lecture 2

Variable:-

Variable is a memory location to save address. Hr variable ka apna koi na koi address hota he jo dusri memory se different hota he.

Note:-

There is no datatype of pointers. Pointers ka size fix hota he, it does not depend on the size of associated datatype, it only depends on architecture. (Addressing space of system) If addressing space is 32 bits = 4 byte then pointer size is also 4 byte.

eg char a = '2';

char * p; (pointer ka size 4 byte ho ga agr addressing space 32 bit = 4 byte hn gi)

Step 1:

// Declare a variable of integer type

int a;

// Initialize it with any integer value

a = 10;

Step 2:

// Store address of variable into another variable

int* p = &a

Step 3:

// display the content of variable declared at Step 1 & Step 2.

cout << a << "\n"; // 10

cout << p;

// address print

Dereference Operator → *p = 20; → To change the content of variable whose address is stored in pointer.

Output:// 20 cout << a << endl;

∴ Display address of pointer :-

cout << &p;

To save address of pointer in another variable:-

asterix for second pointer

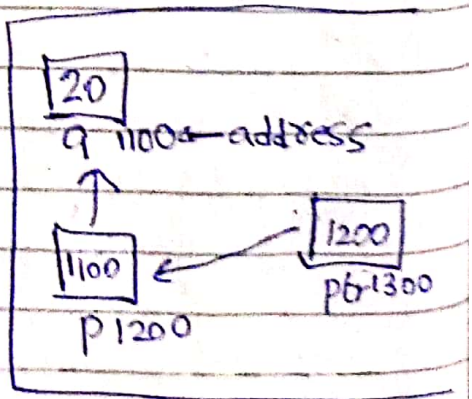
pointer to pointer → int ** ptr = &p;

information of first pointer (p)

** ptr = 25; →

dereference a ko by using address in p.

dereference p ko by using address in ptr



cout<<a<<endl; // output:- 25

Pass by value

```
void func(int a)
{
   a=10;
}
void main()
{
   int a=5;
   cout<<a<<"\n";
   func(a);
   cout<<a<<"\n";
}
```

:. New block bne ga

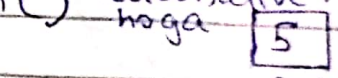
5

Output: 5

Pass by reference

```
void func(int& a)
{
   a=10;
}
void main()
{
   int a=5;
   cout<<a<<"\n";
   func(a);
   cout<<a<<"\n";
}
```

:. New block nhi bne ga blke pehle se mujood block ka alternative name hoga



another name of memory location

5

Output: 10

:. The instance made by built-in datatype is called variable

:. The instance made by user-defined datatype is called object.

* Reference variable:

```
void main()
{
   int a=10;
   int &b;
   b=a;
   int &b=a;
   cout<<b<<"\n"<<a<<"\n"<<&b<<"\n"<<&a;
}
```

Ye invalid he kiun ke hm compiler ko new block bnane ko keh rhe jb ke pass by reference me aesa nhi hota.

X invalid

valid

output: 10
10
1100
1100

Lecture 3 (22-02-2022)

Classroom Code :- 25at5on

Passing a Pointer

By value

parameters

```
void func(int* p)
{
   *p = 20;
   p = 20;
}
void main()
{
   int a=10;
   cout<<a<<endl;
   int *ptr = &a
   func(ptr);
   cout<<*ptr<<endl;
} //output : 20 (a ka content)
```

20, 1100, p 1300

*p = 20; → search kre ga 1100 address wale block ko aur phir us ka content change kre ga 20 se.

p ki value change kr ke 20 kr do. ← p = 20;

10, 20, a 1100

1100, ptr 1200

argument

∴ There is a typecasting error in it. (p)

∴ local variable ke through original variable (a) ka content change kya.

By Reference

const → error aye gi kiun ke const likhne ka valid synt kota he i.e dono scope me likhte.

```
void func(int* p)
{
   *p = 20;
   p = 20;
}
void main()
{
   int a=10;
   cout<<a<<endl;
   int* ptr = &a;
   func(ptr);
   cout<<*ptr<<endl;
}
```

10, a 1100

1100, ptr 1200

∴ There is a typecasting error in it.

∴ const ko defer nhi kr skte

Typecasting error :- By value aur By reference dono ke function me 1st step me to original variable (a) ke content ko change krte hein but 2nd step me p = 20 me hmne p ko use kr ke ptr ke content ko change kr dia jis se original var (a) aur pointer variable (ptr)

ka link khtm ho jaye ga jis se typecasting ka error ayega. kiun ke hmne int* p likha hua jis ka mtlb pointer to int datatype. By value me to local variable change hota he lekin By reference me pointer (ptr) change ho jaye ga is liye hm const ka function use kre ge ta ke wo parameter original pointer (ptr) ke content ko change na kr ske.

OOP

24-2-2022 Lecture:- 4

Const with pointers

1- Allowed / Valid passing by value

```
void func(const int* p)
{
    na data ko ↵ alter kr skte or na hi pointer ko defer kr skte kiun ke p const he aur wo ye allow nhi kre ga.
}
void main(){
  int * ptr1 = nullptr;
    func(ptr1);
}
```

kr ke var ka content change nhi kr valid

2. Not Allowed / invalid passing by reference

```
void func(const int* & p){

}
void main(){
  int * ptr1 = nullptr;
    func(ptr1);
}
```

Invalid

:: Iscope me const he aur dusre scope me const nhi he -> error

Note:- Pointer store the address of variables which executes at runtime. Pointer link with variable at runtime.

3- Allowed / valid Passing by Reference

→ na data ko alter skte aur na hi defer kr ke original variable ka content change kr skte simple defer kr skte.

```
void func (const int*&p){

}
void main(){
  const int* ptr1 = nullptr;
    func(ptr1);
}
```

Valid (kiun ke dono scope me const he)

Note:

∵ Abort, Break error = No memory location of the variable that is call

nullptr:-

- 0 -> pointer ki base value.
- 0 (zero) ksi ki bhi memory location nhi hota. hm pointer to ~~null~~ zero se initialize ke equal krte he kiun ke zero hexa me bhi zero hota he. lkin zero krne se memory consume hngi aur hm chahte hain ke koi memory consume na ho is liye hm nullptr se pointer ko initialize kre gein.

nullptr ; NULL
↓ ↓
New version Old version

~~nullptr~~ nullptr -> pointer ko khali kr de ga ie us me koi value nhi aye gi aur na hi koi memory consume hngi

Memory
↳ Aik hi memory kbhi stack ki trah behave kregi aur kbhi heap ki trah

Static memory allocation

Dynamic programming
Dynamic memory allocation

Stack
volatile
- compiler remove memory

Heap
non-volatile
- programer remove memory

lecture 5

* Dynamic Memory Allocation:-

1- -> Creation / Allocation of memory at **run time.**
2- -> Dynamic memory allocation is done through **pointers.**
3- -> Dynamic memory always created at **Heap.**
4- -> Deletion of Dynamically created memory is duty of **programmer.**

int a,b,c,d;
if{
c.d-multiplication}
else{ a+b sum}
Ab agr condition asi ho ke hmesha if hi chle to a aur b sirf memory waste krein ge is ko avoid krne ke lye hm run time pr box bnate hain in order to save memory.

-> Creating integer variable statically:

```
int a;
a = 5;
```

-> Creating integer variable dynamically:

```
int* ptr = nullptr; (statically)
ptr = new int ; (dynamically)
```

(statically) -> hm sirf datatype likhte ge
(dynamically) -> Integer type ka box bna do run time pr aur ye hmesha address return kre ga jo ke box ka address ho ga jise create kya he kiun ke is box ka koi naam nhi he. is ke bd ye address pointer me store ho ga jis ki help se hm box ko access kr ske gein.
variable ka naam nhi

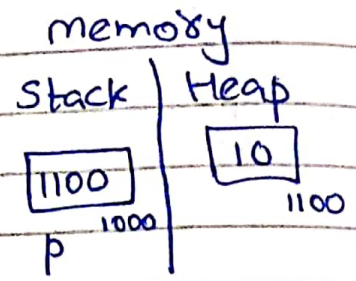
new = keyword

Note:- zaroori nhi jha pointer ho wha dynamic memory ho lkin jha dynamic memory ho gi wha pointer zaroor ho ga address save krne ke lye.

Note:- OOP is also referred as real world programming.

Deletion process:- (How to delete dynamic memory)

```
int* p = nullptr;
p = new int;
    ↓
  keyword
*p = 10;
```




```
delete p;
   ↓
keyword
```

→ (is se pehle ke stack me mujood pointer p ko compiler remove kre hm us pointer ke through heap me mujood dynamic memory ke box ko remove kre ge kiun ke pointer me us ka address he aur hm sirf pointer ke through hi is box ko delete kr skte.

```
p = nullptr;
```

↳ is-se dangling pointer nhi ayega

agr hm ye na bhi likhe to theak he phir garbage value ho gi. re dangling pointer ko avoid krta he.

Dangling Pointer:- Pointer that holds the address of that memory location which does not exist. (jis ki koi base na ho اسے؟)

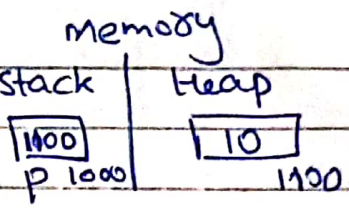
```
int* p = nullptr;
p = new int;
*p = 10;
delete p;
*p = 10;
```

→ Dangling pointer (kiun ke hm jis address ko defer kr rhe uski memory exist nhi krti.)

Note:- Dangling pointer static memory me nhi hota sirf dynamic memory me hota.

Overall syntax:-

```
int* p = nullptr;
p = new int;
*p = 10;
delete p;
p = nullptr;
```



Defect:- memory waste ho rhi pointer ki 4 byte agr system architecture 32 bit he stack me aur int ki 4 byte heap me. memory aik hi hoti bs kbhi stack aur kabhi heap ki trah behave krti.

Note:- Visual studio me hm array ka size user se input nhi krwa skte bt hm ab user define size se array bnaye gein:-

```
int* p = nullptr;
p = new int[5];
```

→ array ki datatype aur size likhe gein
∴ int arr[5];

aur int a = ___

~~int* p~~ p = &a;

OOP

3-3-2022 lecture 6

Creation / Allocation of array on user-defined size

→ Not allowed

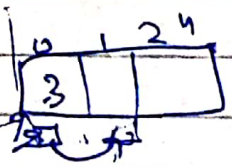
```
void main(){
   int size;
  cout<<"Enter size: ";
     cin>>size;
     int arr[size]; //invalid
}
```

→ Allowed

```
void main(){
    const int size=5;
      int arr[size]; //valid
  //But not user defined
}
```

//Dynamic Allocation of array at user-defined size.

→ Alowed




```
void main(){
  int size;
  cout<<"Enter size:";
     cin>>size;
  int* p=new int[size];
 for(int i=0;i<size;i++){
  cout<<"Enter"<<i+1<<"value:";
     cin>>*(p+i); //valid
    //cin>>p[i]; valid
 }
   detete [ ] p;
    p=nullptr;
} //end of main( )
```

Passing Dynamic Array to function

```
void func(int*arr, int s){
  for(int i=0;i<s;i++){
      cout<<arr[i]<<endl;
  }
}
void main(){
  //(same code as in previous main function)
     func(p,size);
} //end of main()
```

Returning pointer/Address from a function

```
int * func(int s){
   int * arr=new int[s];
  for(int i=0;i<s;i++){
    cout<<"Enter"<<i+1<<"value:";
      cin>>arr[i];
   }
     return arr;
} //end of function
void main(){
      int size;
    cout<<"Enter size:";
     cin>>size;
    int * p=func(size);
for(int i=0;i<size;i++){
   cout<<p[i]<<endl;
}
  delete [] p;
   p=nullptr;
} //end of main()
```

1- Transcript
2- subscript

arr[i] -> index ke through (array input ya output ki)

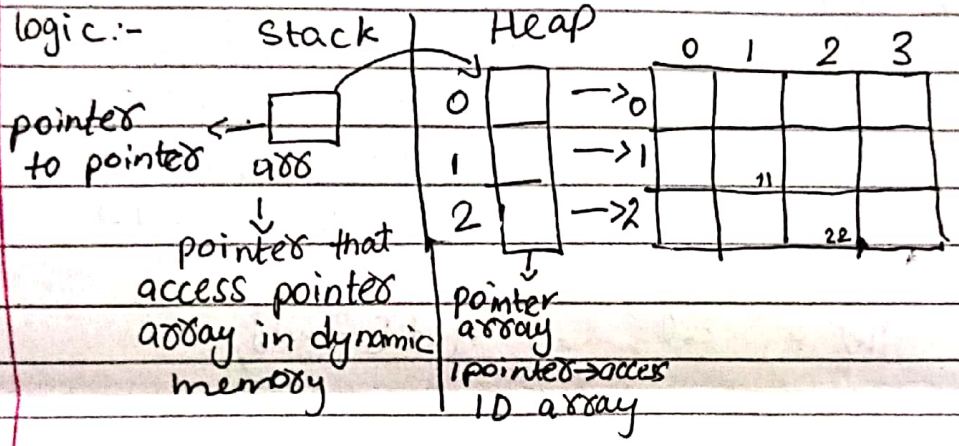
*(arr+i) -> through deference operator

⇒ OOP ∴ 2D array -> multiple 1D array (Rows)

8-3-2022 lecture 7

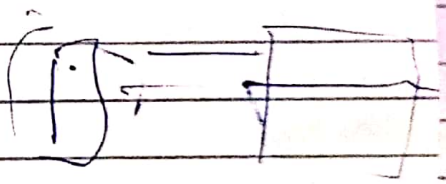
Dynamic Allocation of 2-D Array;

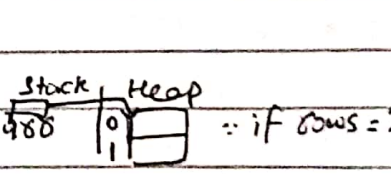
logic:-

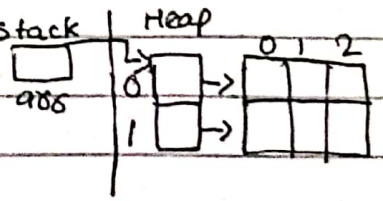



```
void main(){
    int rows, cols;
    cout << "Enter rows: ";
    cin >> rows;
    cout << "Enter Columns: ";
    cin >> cols;
// Creating array of pointers
    int** arr = nullptr;
    arr = new int*[rows];
// Creation of 2D array
    for(int i=0; i<rows; i++){
        arr[i] = new int[cols];
    }
// input values
    for(int i=0; i<rows; i++){
        for(int j=0; j<cols; j++){
    cout << "Enter value at" << i << j << "index:";
            cin >> arr[i][j];
            // cin >> *((*(arr+i))+j); -> Try it on Visual studio
    }
// end of external loops
// output values
```

Another syntax of cin>>arr[i][j];





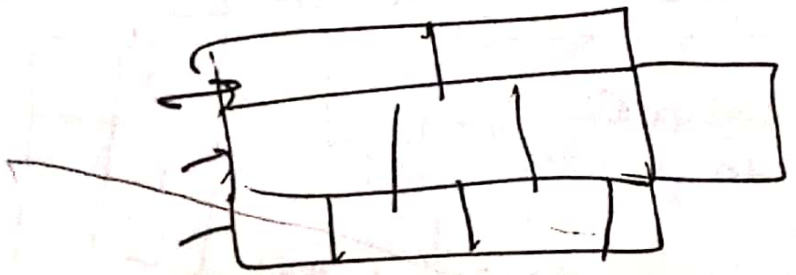


// Deletion of 2D array:-

```
for(int i=0 ; i<rows; i++)
{
    delete [ ] arr[i];
    arr[i] = nullptr;
}
delete [ ] arr;
arr = nullptr;
}
```

rows is liye kiun ke jitni rows thi utni hi pointer ki 1D array bhi thi.

★ column ki arr



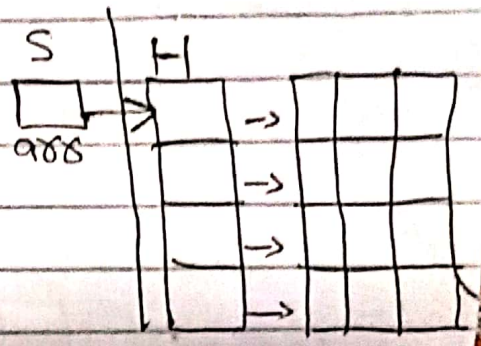
```
int * col = new int[rows];
```

OOP

10-3-22 lecture 8

* Accessing 2-D Dynamic Array using deference operator:

```
for (int i=0 ;i<rows ; i++)
 {
   for (int j=0; j<cols ; j++)
  {
    cin>> *((*(arr +i))+j)
   }
}
```



* Passing and Returning 2-D dynamic array to and from a function

```
int** func( int** arr, int r, int c)
{
  // code segments
}
void main(){
     int** ptr = func( //arguments );
 }
```

Data type

primitive/built in — derived from primary

Abstract / user defined — programmer

Concept of classes:

Class: A blueprint

built in ka instance — variable
create ho ga tabhi hoga

OOP

Type of mid QS:
- int * arr[5];
- arr[i] = new int[5];
- arr[i] = new int[size]; different size

Lecture 9

15-3-2022

=> Abstract Data Types
- -> programmer / user defined data types
- -> Instance / Occurance of Abstract data type is called Object.

Q -> Need ?

Primary
- Primitive datatype -> create instance called variable
- Abstract datatype -> create instance called Object i.e Instances of Abstract data type.

=> OOP -> real world programming

Multiple memory location ko aik box me pack kre gein to wo class ho gi.
Class ⟶ most related -> memory location.

Abstract datatype
class struct

datatype
int a, b;
primitive Instance (variable)

int c = a + b;
↓
no error in + symbol as it is primitive and logic is written in file #includeciostreamz

if there is userdefine datatype
↑
a, b;
c = a + b
↓
there will be a error in it as data is not primitive, it is abstract so we have to write logic of +, first and then it runs

H.W => Difference b/w class & struct. (5.6)

Difference
=> Linking (Dynamic, Statically).

=> Primitive → unlimited instance create krskte
to Abstract ke bhi → unlimited instance create
krskte.

→ Class → blue print
↑
unlimited instances

→ Syntax:- Any name
key-word class MyFirstClass → (writing convention: Camel Case
{ variable ke naam me space nhi hoti to hm hr word ka 1st alphabet capital likhte ge)

// Coding statements

Another method:
my_first_class
or
My_First_Class

};

* Access:-
By default access of classes are private.

* Implementation (Example)

```
class MyFirstClass
{
  int a,b,c;
};
void main()
{
  MyFirstClass obj1;   // object / Instance.
  cout << sizeof(obj1); \\5
  //obj1.a=4;
}
```

data type

Run on visual studio
(obj1 ke andar a search kro usme 4 save kro pr ye nhi ho ga kiun ke by default private he)

Access Modifiers (Access change krne ke tareeqe)

1- Private
2- Protected } keywords
3- Public

OOP ki 1st property:-

1- Hiding/Privacy

OOP

17-3-2022 Lecture 10

Abstract Data type:-

- Abstract Data type → variables not fixed
- Function class me likhe ge kiun ke class ko pta ke wo private he. hm function ke through public kre gein.

```
class MySecondClass
{
  int a,b,c; -> class ke andar jitni bhi memory location ati wo data members hoti
                (variable, array, pointers etc.)
public:
    // Setter/Mutator
  void setA(int val){ -> class ke andar jitne functions ate wo member function hote.
    a=val;
  }
    // Getter/Accessor
  int getA(){
    return a;}

};//end of class
```

```
void main(){
  mySecondClass obj1;
  obj1.setA(4);
  cout<<obj1.getA();
}//end of main()
```

→ show kre ga sb function jo Public me he

mid 1:=

⇒ dry run, pointer to pointer, dynamic 2D array

Set All krna ho to → Uthe hi parameters pass kre gein jithe variable hmne class me bnaye hnge.

conventional hunga agr hm setAll ke sath bhi sath sb ka setter alg alg bnade

Agr sb ko alg alg set krna to → 3 setters bnaye gein

Scenario: We have to create 1D dynamic array using class.

```
Class DynamicArray{
    int size;
    int* ptr=nullptr;
public:
void setsize(int n){
    size=n
  }
int getsize(){
    return size;
  }
void allocateArray(){
    ptr=new int[size];
  }
```

→ private me kiun ke ptr jis me hmne store kya wo private tha

```
void inputValues( ){
for(int i=0; i<size;i++){
    cout<<"Enter"<<i<<"value: ";
        cin>>ptr[i];
  }
} // end of inputValue
 // Display values (write code yourself)
 void deAllocate(){
    delete [ ] ptr;
       ptr=nullptr;
 }
}; // end of class

void main( ){
  DynamicArray object;
    int s;
    cout<<"Enter size: ";
      cin>>s;
    object.setSize(s);
    object.allocateArray( );
    object.inputArray( );
    object.display( );
    object.deAllocate( );
} // end of main( )


void setsize(int n)
  {
    size=n;
  } ptr=new int [size];
```

allocateArray() bnane ki zaroorat nhi hogi.

OOP

29-3-22 lecture 11

Constructor (Imp topic)

Jb aik object create hota to background me function (constructor) call hota he aur ye function sirf or sirf usi wqt call hoga jb obj create hoga.
Ye built-in bhi ho skta

Constructor
↓
Special type of Function

- It has no return type (Ye void se alg he void return type ka mtlb hota ke kuch return mb kro.)
- Name of constructor should be exactly same as class name.
- We don't need to call constructor function specifically.
- Constructor will be automatically called whenever an object is created.

Type of Constructor

- Parameter list empty → Non-parameterized
  - Default
  - Programmer-defined
- Parameterized
  - Programmer-defined.

because data members are private in class
∴ Programmer built constructor in class and it's acess is public (as main is public).
and constructor is called in main when object is created.

```
class A
{
  int n1,n2,n3; -> data members
  public:
    // constructor
    A()
    {
      n1=5;
      n2=6;
      n3=7;
      cout<<"Non-parameterized programmer-defined Constructor";
    }
    // Setters & Getters for n1,n2,n3 (In order to further change values of n1, n2 and n3.)
};// end of class

void main()
{
  A a; (do option hein ya to default constructor chle ya programmer define is case me programmer define chle ga kiun ke uski preference zyada he.)
```

for checking purpose

∴ Constructor ke function ko khi bhi likh skte hein but within the scope of class.

∴ Function overloading : (3 steps for Function overloading)

(1) no. of parameters
(2) change datatype of parameters
(3) order of parameters change

```
class A
{
    int n1, n2, n3;
 public:
        //constructor
  A()              -> constructor overloading
  {
     n1 = 5;
     n2 = 6;
     n3 = 7;
 cout<<"Non-parameterized programmer-defined constructor\n";
}
//Setters and getters for n1, n2, n3
// Parameterized Constructor
A (int val1 , int val2, int val3)  -> constructor overloading
 {
     n1 = val1;
     n2 = val2;
      n3 = val3;
  cout<<"Parameterized Constructor\n";
  }
}; //end of class
void main(){
 int v1, v2, v3;
 cout<<"Enter three values: ";
   cin>>v1>>v2>>v3;
   A obj1(v1, v2, v3); -> Parameterized function ko values pass krne ka tareeqa.
   A obj2; => A obj3();
     -> non-parameterized wala -> By default wala chle ga
```

Error information: when constructor is not available i.e you pass three arguments but make constructor function of 2 parameters.

OOP

31-3-22 lec 12

Constructor

// Parameterize + Non-Parameterized Constructor

Code Source: Lec 11

```
A(int val1=5, int val2=6, int val3=7)
{
    n1=val1;
    n2=val2;
    n3=val3;
cout<<" parameterized + Non Parameterized Constructor\n";
}
void main()
{
  A obj1(10,12,14);
  A obj2;
  A obj3(); // ??
}
```

→ agr 2 arguments pass ho aur parameters yun likhe hn i.e int val1=5, int val2=6, int val3=7 to error ajega.

→ jb ye run hoga to parameters ki value change hngi i.e val1=10, val2=12, val3=14 if we write it as A obj1(10,12) there will be no error values become val1=10, val2=12, val3=7

→ jb ye run hn ga to values change nhi ho gi i.e val1=5, val2=6, val3=7.

H.W (upper wale dono (obj1, obj2) ko comment kr ke ye line run kro)

*Scope Resolution Operator (Regarding Classes)

→ Symbol: ::

→ Uses to define member function out-side the scope of class.

→ Uses to initialize static data member(s).

e.g:

```
int x=5;
void main(){
  int x=6;
  cout<<x // 6
          local variable
  cout<<::x // 5
          global variable
```

```
class A
{
  int n1, n2, n3;
public:
  A(int val1=5, int val2=6, int val3=7);
    void setN1(int i);
     int getN1();
};  // end of class
```

class name → to access and to change private data

```
A::A(int val1=5, int val2=6, int val3=7)
{
  n1=val1;
  n2=val2;
  n3=val3;
}
```

hm ye functions kahi pr bhi bna skte main ke bd bhi

In case of return type function we write return type → class name

```
void A::setN1(int i)
{
  n1=i;
}
void main()
{
  ~~~~
  ~~~~
  ~~~~
}
```

General Syntax

```
Return Type classname::Function name(...)
{
  .....
  .....
}
```

⇒ Singleton pattern :- Poore scope me aik class ka aik hi object create ho skta. Wese aik class me hm unlimited object create kr skte hein.

Destructor :-

→ A special type of function
→ Name of Destructor should be exactly same as class name.
→ will be automatically called when the scope/life of object is going to terminate/finish.

e.g

Code Source : Lec 11

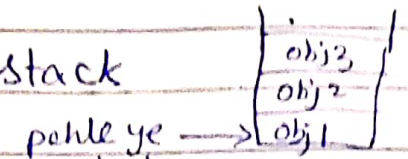
```
class A
{
public:
        // Same code
        // Destructor
        ~A()        // Tilde sign (Distinguish Destructor from constructor)
        {
          cout<<" Destructor \n";
        }
}; // end of class
void main()
{
  A obj1;
}
```

* Destructor sirf non parameterized hoga parameterized me nhi.

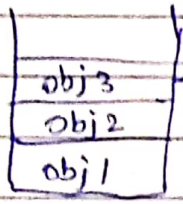
destructor of C

Mid 2: Imp

object order → save in stack



Destructor :-



→ pehle ye delete hunga aur phir neeche wale i.e reverse order.

jis order me object create hote usse reverse me object destruct hote. because object order is save in stack

3 classes bni 3no ka alg alg object bne ga. Agr hmne pehle A class ka obj1 bnaya, phir B class ka obj2 bnaya aur end pr C class

OOP

5-4-22 Lec 13

Copy Constructor

Constructor that will be automatically called whenever a newly created object is being initialized with an existing object.

∴ Copy constructor is also parameterized

Example:-

```
class DynamicMemory
{
    int size;
    int * arr = nullptr;
public:
    // constructor
    DynamicMemory(int n=5)
    {
        size = n;
        arr = new int[size]
        cout << "constructor\n";
    }
for(int i=0; i<size; i++)
    {
        arr[i] = obj.arr[i];
    }
} //end of copy constructor
    // copy constructor
DynamicMemory(const DynamicMemory & obj)
    {
size and arr are obj3 components { size = obj.size;
                                   arr = new int [size];
```

we are using obj 2 if only obj is written it use obj3. i.e obj.size = obj2size size = obj3 size

```
// input function
 void input()
// Display function
   void display
```

nullptr

```
// Destructor
  ~DynamicMemory()
  {
    if(arr1 != nullptr)
    {
      delete [] arr;
      arr = nullptr;
    }
  }
}// end of class

void main(){
  DynamicMemory obj1;
  DynamicMemory obj2(10);
  DynamicMemory obj3 = obj2; or DynamicMemory obj3(obj2);
    obj1.display();
    obj2.display();
    obj3.display();
  }
```

* Programmer define constructor → deep copy → above program → value copy krke memory create krta he.

Default copy constructor :- / Shallow copy cannot creat memory → Header / DynamicMemory(const DynamicMemory & obj) signature same new memory create krne ki capability nhi rkhta. bit value copy krta. 1st variable of obj2 is copy in 1st variable of obj3 i.e size of obj2 into obj3. 2nd variable of obj2 is copy in 2nd variable of obj3 ab pointer arr me dynamic array ke 0 index ka adress he same whi address obj3 ke pointer arr me store hoga. jis se obj2 ki arr me hi changing ho jaye gi jb user define copy constructor me obj3 ke pointer me dynamic array ke 0 index ka new address copy hoga i.e new arr bne gi.

=> Operator overloading :-

DynamicMemory obj3; -> Non-parameterized constructor chle ga

obj3 = obj2; -> is pr koi constructor nhi chle ga.

OOP

7-4-22 lec 14

Operator Overloading :-

-> Assignment Operator

Symbol : =

-> Default copy constructor does shallow copy.

So if we have to do deep copy, we must overload assignment operator.

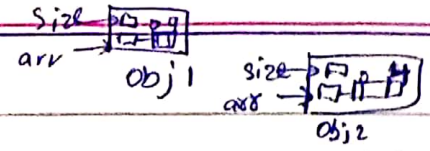
Code Source : lect 13

keyword -> class object -> jis operator ka logic likhna h -> programmer define operator.

```
void DynamicMemory ::Operator=(const DynamicMemory& obj
{
    if(arr != nullptr)
    {
        delete [] arr;
        arr = nullptr;
    }
    size = obj.size;
    arr = new int[size];
  for(int i=0; i<size; i++){
        arr[i] = obj.arr[i];
    }
} //end of operator

void main()
{
```

To delete current memory of obj2.

int c = a + b




```
DynamicMemory obj1(10);
DynamicMemory obj2;
   obj2=obj1;  -> Assignment operator default primitive function he to is waja se ye shallow copy kr rha mgr hmpe deep copy krwan
// DynamicMemory obj3=obj2; //copy constructor
   -> or obj2.operator(obj1);
}
```

* Returning an object from a function:

```
class A {
    int a;                                   dynamic -> = deep
public:
    A(int n=5) {
        a=n
        cout<<"Constructor\n";
    }
    ~A() {
        cout<<"Destructor of"<<a<<endl;
    }                  -> (A obj) object create -> constructor
A getobject(A& obj) {  -> object return kre ga to object ki returntype A means class ka naam hnga.
        A=temp;
        temp.a=obj.a;
        return temp;
    }
};
void main(){                                  Output          -> temp
    A obj1;                                   Destructor of 10
    A obj2(10);                               Destructor of 10   -> obj
    A obj3;                                   Destructor of 10
obj3=obj1.getobject(obj2)                     Destructor of 5
}//end of main class
```

Gadies:- Starting out with C++ -> operator overloading.
L> ch 14

OOP

12-4-22

Lec 15
L> Related to mid 2 output and long Quest
L> static
L> global

We will cover following:-
1- Dynamic object(s) creation
2- this pointer "this" is a special type of pointer that is attached to every object and pointing itself
3- Operator overloading
   L> Increment (++)/ Decreement(--) operator (pre+post)
4- More about copy constructor

*Code:

```
class learn
{
  int a;
public:
   learn(int a=5){
     this->a=a;
    cout<<"Constructor\n";
   }
   learn(const learn& l)
   {
     a=l.a;
    cout<<"Copy Constructor\n";
   }
learn func(learn l1, learn l2, learn& l3)
   {
     static learn x;
      x.a=l3.a;
       return x;
   }
learn operator++()
{
```

Copy constructor chle ga

New objects bne ga aur bnte sath arguments se aan wala data, copy ho jaye g

Static -> remain lively in all the program

copy constructor chle ga

Function se object return kre ga object fresh new memory location bnti aur us memory me x copy ho ga.

a+b → void
single

a+b+c → return
class
multiple

a = a
class ke a me receive are wale a

Pointer create
Learn* p = new Learn(20)
object create

(*p → deference p
poora object

this → pointer of every object
hidden pointer

(*p). func
↳ p → func( );
arrow static me nhi bnta

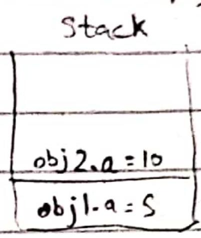
```
Learn temp = *this;
    ++a;           → object poora ka poora return hoga.
 //return *this;   → to hm likhe gein = return temp;
 }
```

copy return hogi bt hmne old copy return krwai

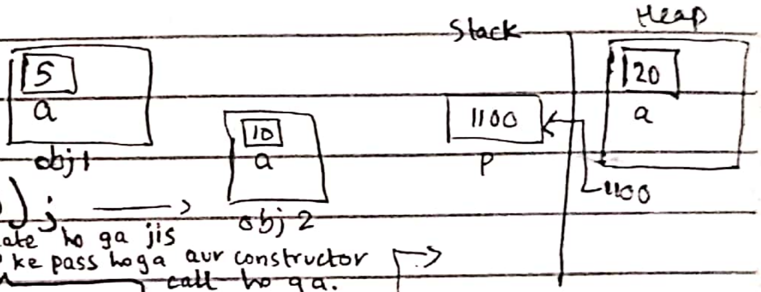
post increment → Uniary operator: no arguments.

```
learn operator++ (int)
 {
    a++;
    return *this;
 }
}; //end of class
```

(int) ↳ ye sirf post me likhe gein dummy parameter he ye kuch accept nhi kre ga just aik neeshani he ke ye post wala he.




```
void main()
{  → Constructor create
   learn obj1;
   learn obj2(10);
```



Pointer create hoga koi object nhi

object create ho ga jis ka address p ke pass hoga aur constructor call ho ga.

```
   learn*p = new learn(20);
```

Same any one:

```
   (*p).fun(        );
   p → func(        );
```

(*p).fun → *p ke sath parenthesis zroori he wrna operator preference ka error ayega. (.) ka preference zyada ho ga. Another method to write same statement is p → func( );

p → func: arrow static me nhi bnta

```
   obj2 = obj1++;
   obj2 = ++obj1;
```

Console:
Constructor (obj1)
Constructor (obj2)
Constructor (dynamic)

arguments → function ko as a parameter pass kre gein dur agr parameters pr copy constructor call ho rha aur aik parameter by reference paas hoa us ka koi constructor nhi chle ga. Stack me reverse order pr memory bne gi.

```
learn func(learn a, learn b, learn c)
{
    static learn x;
    x.a = c.a;
    return x;
}
```

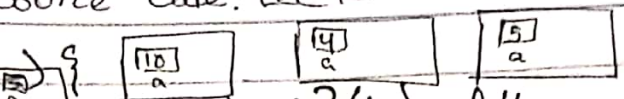
OOP

14-4-22

Lecture 16

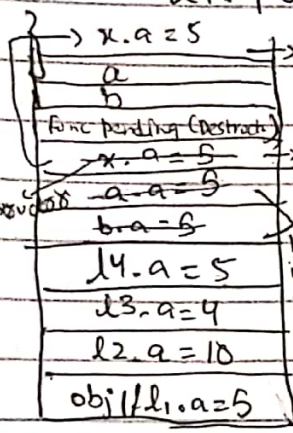
Source code: Lec 15

```
void main()
{
    learn l1, l2(10), l3(4), l4;
    l1.func(l2, l3, l4);
    l1.func(l2, l3, l4);
}
```



return x ka destructor is line ke execute hone ke bd chle ga.

Mid 2 output diy.



is br static(x) nhi bne ga pehle wala use hoga jb pehle wale ko use kre ga to pehle ka destructor chl jaye ga

Static remove nhi hoga

jb function khtm hoga inka destructor chle ga

Constructor (l1)
Constructor (l2)
Constructor (l3)
Constructor (l4)
Constructor (b) → function
Constructor (a) → function

Destructor	x
Destructor	a
//	b
//	l4
//	l3
//	l2
//	l1

* Friend function and friend classes:-

friend: keyword that is used by a class to give right to access its private data member(s) to a simple function or to a function inside a class or to a whole class.

→friend function:

```
void display(A);
class A {
    int a;
    friend void display(A);
};
```

→ // Forward declaration of Fuction.
→ function ka header (signature) jo class ke bahir bna hua hmne (A obj) ki jgah sirf (A) likha kiun ke is me hm sirf define krein gein use nhi jb use krna ho to object bnana prta he. is se error nhi ayega.

->Recursion

```
void display(A obj)
{
  cout<<obj.a<<endl;
}
```

->friend function: (Function inside other class)

```
class A; // Forward Declaration of class
class B
{
  int b;
  public:
        void display(A); -> signature.
}; // end of B class.

class A    B use few features of A.
{
  int a;
  friend void B::display(A);
}; // end of class A

void B::display(A obj)
{
  cout<<obj.a<<endl;
}
```

write this class in place of upper class

->Friend classes.

```
class A
{
  int a;
  friend class B; -> class friendship. A ko bta ya ke B uska friend he. B, A ki chee zon ko use kr skta bt A B ke private members ko nhi kr skta.
}; // end of class A
```

→ Relationship among classes

1- Dependency → use-a keyword → upper concept ka relationship dependency he. It is a weak relationship-
2- Association
3- Aggregation
4- Composition
5- Inheritance

OOP

19-4-22 lecture 17

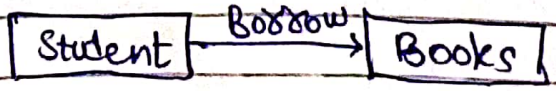
→Relationship among classes :-

1- Dependency (use-a) .....>
2- Association (use-a) ——>
3- Aggregation (has-a) ——◇
4- Composition (whole part) ——◆
5- Inheritance (is-a) ——▷

2. Association:

→ Weak Relationship
→ No ownership is involved.
→ Object lifespan is independent.
→ can be one-one, one-many, many-many

e.g

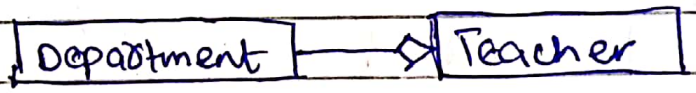


```
class Student
{
    int sId;
 int * BorrowedBook;
public:
      void BorrowBook (const int &id);
      void return_Book (const int &id);
};  // end of student class
```

3- Aggregation:

-> Weak Relationship
-> Ownership is involved
-> Independent life span.
-> can be one-one, one-many

e.g:



Code Example:-

```
let
    classes= A, B
class B
  {
    int b;
public:
```

```
        B(int n=5);
};

class A
{
    int a;
    B* obj = 0;
public:
        A(int n=3);
    void addB()
    {
      obj = new B;
    }
    void deleteB()
    {
        delete obj;
        obj = 0;
    }
    ~A();
};  // end of A class
```

4- Composition:

→ strong relationship
→ ownership is involved.
→ Dependent life span
→ can be one-one, one-many.

e.g.:-

Code example:

```
class B
 {
    // same as previous
};

class A
 {
    int a;
    B obj;
// public:
    // constructors
};
```

OOP

lec 18

: friend se private member functions bhi access hote hein

21-4-22

↳ privacy maintain krta

→ **Inheritance** :- A new class is based on existing class. Now class inherits data member(s) and member function(s) except constructor and destructor of class it is based on

→ **Generalization and Specification**:-

Dog ↓ Poddle

→ Poddle **is-a** Dog
→ Ali **is-a** Human
→ Triangle **is-a** Shape

→ Inheritance has **is-a** relationship.

Base class → when relationship create another name of class Parent

```
class Parent
{
    int p;
public:
    Parent(int n=5)
{
    p=n;
cout << "I am Parent\n";
}
void display()
{
```

pl. cl.

→ dependency
→ association
→ inheritance

```
    cout << "Display inside Parent\n";
  }
  ~Parent()
  {
    cout << "Destructor of Parent\n";
  }
};  // end of Parent class
```

Derived class → when relationship create another name of child class

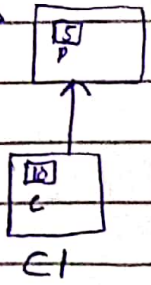
```
class Child : public Parent
{
  int c;
public:
    Child(int i=10)
    {
      c=i;
    cout << "I am Child\n";
    }
  ~Child()
  {
    cout << "Destructor of Child\n";
  }
};  // end of child class

void main()
{
  Parent pt;
  // pl
  Child cl;
  // cl
}
```

keyword — class name

yha se parent class pr jump kre ga is ka constructor chlne se pehle parent class ka constructor chle ga.

Agr hm ab parent class ke constructor ki value change krni he to hm hidden code ko modify kre ge → Child(int i=10):Parent(8)

Hidden code



→ child class ka constructor chlne se pehle parent class ka constructor chle ga.

destructor → child class
destructor → parent class
display → parent class

OOP

lecture 19
April 26, 2022

→ Access Specifiers / Modifiers:
- → private
- → protected
- → public

{ Ch #15
→ Tony adis
→ starting out with C++

→ Protected access

protected member of base class are like private members but derived class may access the protected members.

Code
Example:

```
class A
{
    int x;
protected:
    int y;
public:
    int z;
};

class B : public A
{
public:
    void display ( )
{
    y = 10;
```

```
    cout << y << endl;
  }
};  // end of B class
```

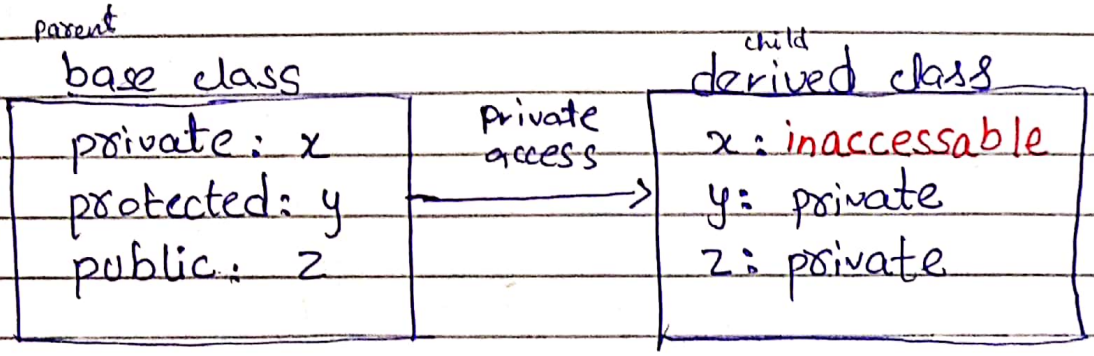
→ Access Specifiers with base class

e.g:

```
                1) Private
class Derived : 2) Protected  Base
                3) public
```

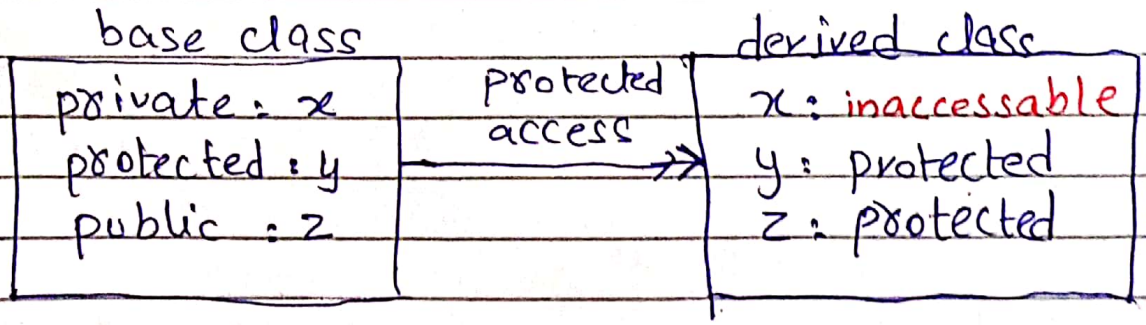
1) Private Access:

e.g class Derived : Private Base
class B : Private A



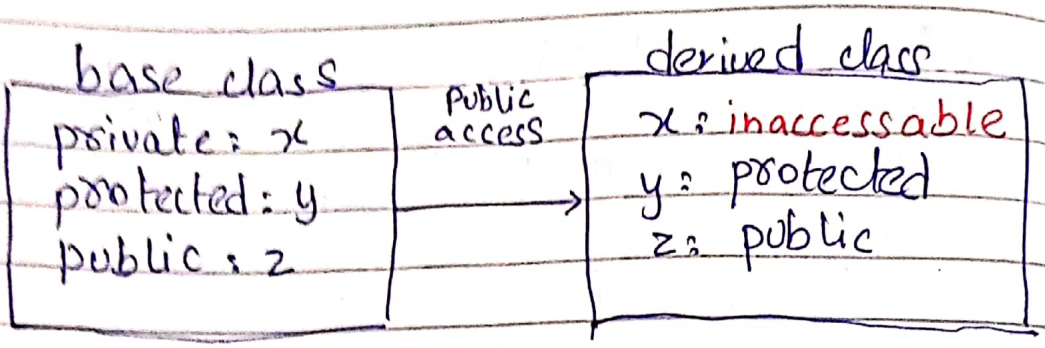
2) protected Access:

e.g class Derived : Protected Base
class B : protected A



3) Public access:

e.g class Derived : public Base
class B : public A



-> Passing values to base constructor

e.g. :

Case: Agr derived class ka object bn rha ho to

: derived class ke parameters ko Base class ke inputs de skte.

-> Derived (int x, int y) : Base ( ) -> Base class ka constructor
{

Jm is me following cheezein likh skte.
- const / literals
- parameter of derived constructor
- global/static variable

}

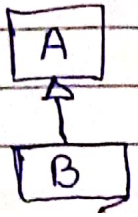
-> Notation:

```
class A
{
};
```

by default access is ~~private~~

```
class B : A
{
};
```

: diagram aa jaye aur program implement krna ho concept necessary.



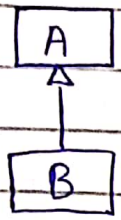
child B inherits A parent

: arrow towards parent.

* Types of inheritance

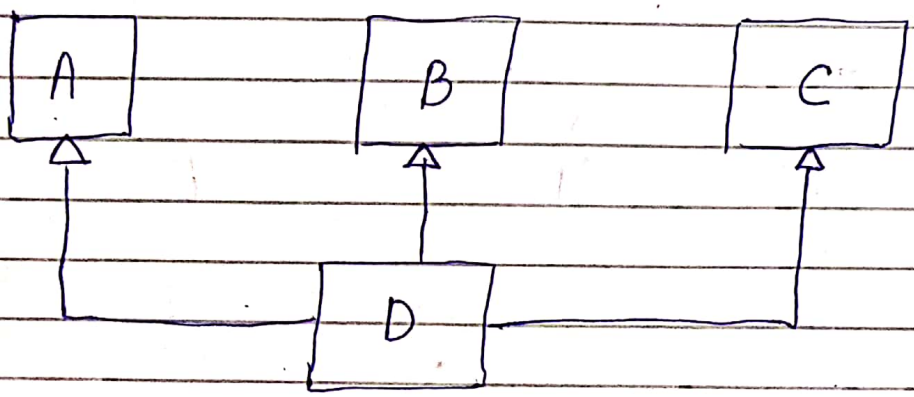
∵ C : A, public B
C class ke do parents with A private access with B public access

① Single Inheritance:

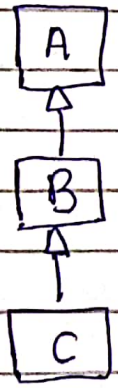


② Multiple Inheritance:

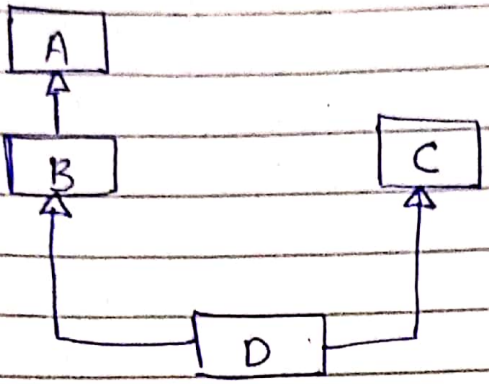
1 child → multiple parents



③ Multi-level Inheritance:



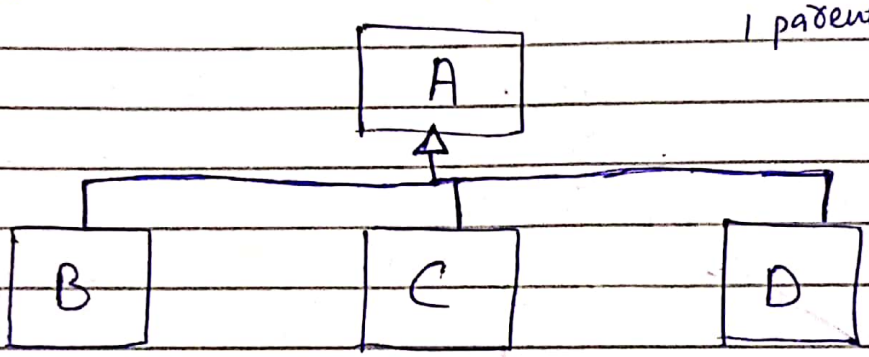
(4) Hybrid Inheritance::



D = child
D ke do parents
different noyiat ke
B A ka child he

(5) Hierarchical Inheritance:

1 parent = multiple child



OOP

28-4-22 Lec 20

→ If a class has multiple Parents

OR

Multiple Inheritance

```
class A
{
public:
    A()
    {
        cout<< "A() \n";
    }
```

```
    ~A(){};
};

class B
{
public:
    B(){
    cout<<"B()\n";
    }
    ~B();
};
```

left to right order

```
class C : public A, protected B
{
public:
    C(){
    cout<<"C()\n";
    ~C();
};//end of class

void main()
{
    C obj1;
    A obj2;
}
```

Hidden code ↑ Idher se jump kre ga aur parent class ka constructor chle ga in left to right order.

Usually, is step pr constructor chlta he. bt use se pehle wo sb data members ke object create krta he.

C class ke constructor pr move kre ga

Console:

A()
B()
C()
A()

Destructor

A.obj2
C.obj1
B()
A()

First A.obj2 remove

→ Redefining Base class function into derived class

```
class One
{
public:
        One();
        ~One();
    void display() {                    — function redefinition
        cout<<"Parent's display\n";
    }
};

class Two : public One
{
public:
        Two();
        ~Two();
  // redefining display
    void display()                      — function redefinition
    {
        cout<<"Child's display\n";
    }
}; // end of Two class

void main()
{
    One a1;
    a1.display(); // Parent
    Two a2;
```

having relationship — different scope me function ka same naam ho

Situation: child aur parent me same naam ka function ho to
↳ C++ → No error → local → child ka display

```
a2.display();  // child
a2.One::display(); // Parent
```
↳ In order to run display function of parent using child object.

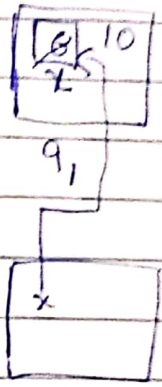
Console:
```
One()
Parent's display
One()
Two()
Child's display
Parent's display
```

→ Static data member(s) and static member function(s)

```
int A::x = 0;   → Syntax to initialize static data member
```

```
class A
{
  static int x;
public:
  A(int n=5);
 ~A();
  static void display()
  {
    cout<<x<<endl;
  }
  void test()
  {
    cout<<"test()\n";
  }
};// end of A class
```
↳ static int x = 0; → error aye ga yha initialize nhi kr skte

```
void main()
{
  A a1(6);
  a1.display();
  A a2(10);
  a1.display;
  a2.display;
}
```



Console
```
6   (1st a1.display)
10  (2nd a1.display)
10  (a2.display)
```

∴ Static member function object create kye bagair bhi call ho skte. → also with object creation

∴ Static member function: can be called without object creation.

```
A::display();
```
→ initialize nhi hota.

∴ static data member/variable me default zero hota he. hm jb bhi static data member bnaye gein to static function me create hota he without creation means constructor nhi chle ga to x (data member) initialize hi nhi hoa) me hm x cout krwa rhe wo uninitialize he to error aye ga. (data member ko initialize krna zroori he agr static hmne static function call krna.

aik class me new object create nhi pehle se mujood class ka obj hi use kre gein. | dynamic (may or may not) | aik class ka scope khtm hone se dusri ka khtm nhi ho ga. → Static [Scope aik class ke khtm hote hi dusri ka scope bhi khtm hota he.

→ Association, Aggregation, Composition

```
class A
{
   public: int n=5
        A (↑);
        ~A();
   };
class B
 {
     int b;
     A a1; -> composition      A a1(10); -> check
 public:
     B(A obj) -> Association    -> copy constructor chle ga, is case me default chle ga (shallow copy)
     {
       cout << "B()\n";
     }
  };
void main()
 {
    A object;
    B b(object); - B ka constructor chlao.
```

data members me jaye ga wha A a; likha to A ka constructor is se pehle chl jye ga

Check: 8

→ B(A obj): a1(7)

class ke andar se dusri class ke constructor ko value paas krni

Gaddie's ch 15

OOP

10-5-22 lec 21

many form/behaviours

Polymorphism

```
class A
{
public:
    A() {
cout << "Parent's Constructor \n";
}
 ~A() {
   cout << "Parent's Destructor \n";
  } // end of destructor
virtual void display()
  {
    cout << " display inside parent class\n";
  }
}; //end of A class

class B: public A
{
public:
   B()
   {
    cout << "Child's constructor \n";
   }
 ~B()
   {
    cout << "child's Destructor \n";
 }
```

in case if there is polymorphism and function redefinition.

↑ means dobara jas aur check kro ke runtime pr decision change ho ga.

object kis ka bna is use

B -> object create
bt hm use nhi kr sakte

```
void display ()
{
  cout<<"display inside child class\n";
} // end of display
}; // end of B class
;
void main()
{
```

static binding:
```
B b1;
b1.display();
```

virtual keyword dynamic binding:
```
A* obj = 0;   -> pointer to parent
obj = new B;  -> child reference store in pointer
```
Valid: Polymorphism relationship necessary

```
obj -> display();
or
(*obj).display();
```
case: jb virtual keyword na ga ho → function redefinition jb relationship hota he to ye function parent ke logically child wala chlna chahye tha ↓ function parent wala chle ga jb

limitation:- Ab agr parent me display naam ka function na ho to is line pr error aye ga. 1st limitation is that ke hm B ka object create krskte ya uska reference store krskte bt hm child ko use nhi kr skte.

jb virtual lg jae to child ka display function chle ga. Agr child ke display ke sath virtual lgaye to parent (A) ka display function chle ga. Parent ke sath virtual prhe ga to child ke sath bhi lg jae ga.

∵ different scope relationship me dono function bt parent wala function virtual ho to:

Function overriding. (Parent function -> virtual)

Note:-

virtual A -> Top parent
↑
B
↑
C

```
A* ptr = new C;
ptr -> display();
```
∵ C ka display function chle ga. Aur agr C me display na to B ka display chle ga.

* Binding:-
  └> Calling correct function on function call

* Static Binding:- (Function redifinition)
  Binding at compile time

* Dynamic Binding:- (Function overriding)
  Binding at run time

OOP

12-5-22

Lecture 22

Polymorphism

Source Code:

lecture # 21

Deletion of Dynamically created object

```
void main()
{
    A* obj = 0;
    obj = new B;
    obj->display;
    // deletion of dynamic object
    delete obj;
    obj = 0;
}
```

Note

```
virtual ~A(){
}
~B(){
}
```

because it is dynamic we have to control deletion parent ka destructor ko virtual kre ge ta ke shi order me deletion ho. pehle B ka delete ga phir wapis aa kr A ka delete ho ga.

* Polymorphism and virtual function(s)-part 2
  Adding a function definition before main() and after declaration of classes.

3 cases: ① jb parent aur child dono me same naam ka function he to jis ka obj create hua uska function chle ga. Parent chle ga.

② Agr function parent me he child me nhi to parent wala hi chle ga. parent=virtual=child me search kre ga nhi mile ga to wapis parent me jaye ga.

③ Agr function parent me na ho child me ho to error aye ga. We child ke andar ja kr dekhe ga hi nhi.

Source Code: lec 21

```
void show(const A& obj)
{
    obj.display();
}

void main()
{
    B b;
    show(b);
}
```

1st combination ↑ child ka reference parent pkr skta he.

2nd combination
```
void show(A* obj)
```
↳ valid

```
main:
B b;
show(&b);
```

3rd combination
```
void show(B* obj)
```
↳ Error: child parent ka reference nhi pkr skta.

```
Main
A a;
show(&a);
```

4th combination
```
void show(A obj)
```
↳ Error: By value, No reference

```
main:
B b;
show(b);
```

OOP

17-5-22 lec 23

Polymorphism - override and final keywords

```
class Parent
{
public:
  // constructor
    Parent();
  // destructor
  virtual ~Parent();
  virtual void display()
  {
    cout<<"display inside Parent\n";
  }
```

: final → func nazeed overrides nhi ho skta i.e limit the scope

```
virtual void test (int n)
  {
    cout << "test inside Parent \n";
  }
};

class child: public Parent
{
   public:
      // constructor
        child ();
      // destructor
      ~ child ();
 void display ( )
   {
      cout << "display inside child \n";
   }
 void test (double n) override
   {
    cout << "test inside child \n";
   }
};

void main ( )
  {
    Parent obj 1;
    Parent *obj 2 = 0;
    obj = new child;
      obj -> display ();
      obj -> test (10);
```

(different from parent)

(keyword (To check wether the func override successfully or not))

```
void test (int n) final
{
cout<<"test inside child\n";
}
```

(Parent ka test func chle ga kiun ke parent child ka signature different he parent → int child → double to dono me ja skta bt preference parent ke func ko ho gi.)

```
    delete obj;
    obj = 0;
}
```

Polymorphism - Abstract base class(s) and pure virtual function(s)

Abstract base class(s)

↳ base class(s) that contain atleast one pure virtual function.

Pure virtual function(s) → Compulsion:- Parent me pure virtual func likha ho to child me override krna zaroori he

↳ virtual function(s) that are declared inside base class but not defined in it. Child class has to override that pure virtual function inside its scope.

```
class Parent
{
  public:
    // constructor
      Parent();
    // destructor
    virtual ~Parent();
```

```
virtual void display ()
 {
  cout<< "display inside Parent\n";
 }
// Pure virtual function
virtual void okay () = 0;
};

class child : public Parent
 {
  public:
    // constructor
     child();
   // destructor
   ~child();
void display()
{
  cout<<"display inside child\n";
 }
void okay()
{
  cout<<"okay inside child\n";
   }
};

void main()
{
   Parent obj1; - error
   Parent* obj2 = 0;
   obj2 = new child;
```

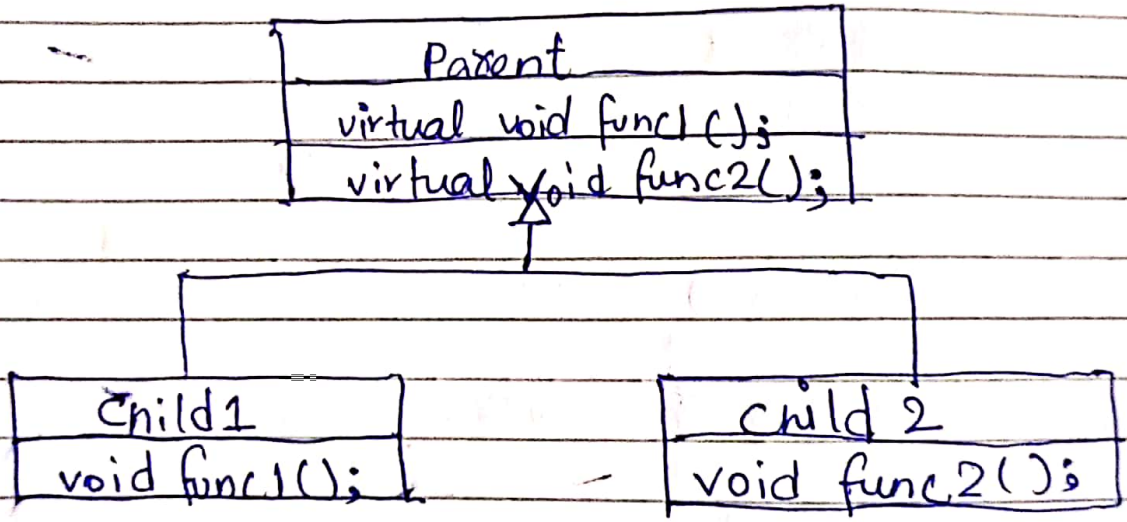
Abstract base class ka instance/obj create hi nhi ho skta.

```
    obj2->display();
obj2 -> okay();
    delete obj2;
    obj = 0;
}
```

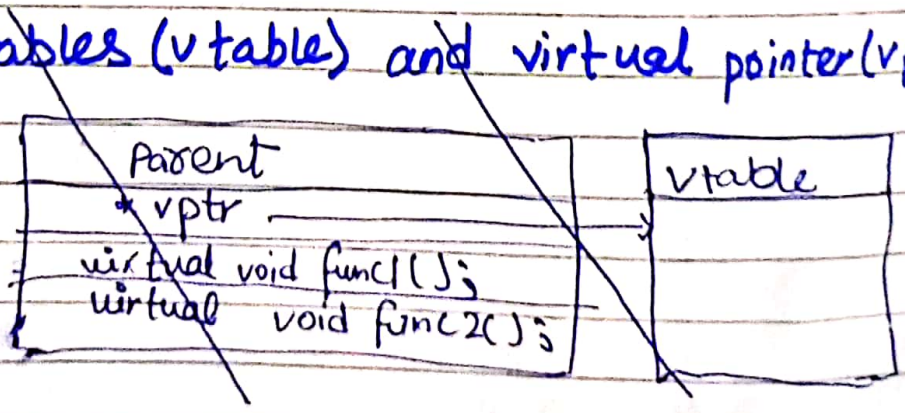
OOP

19-5-22 Lec 24

→ Virtual tables and virtual pointers

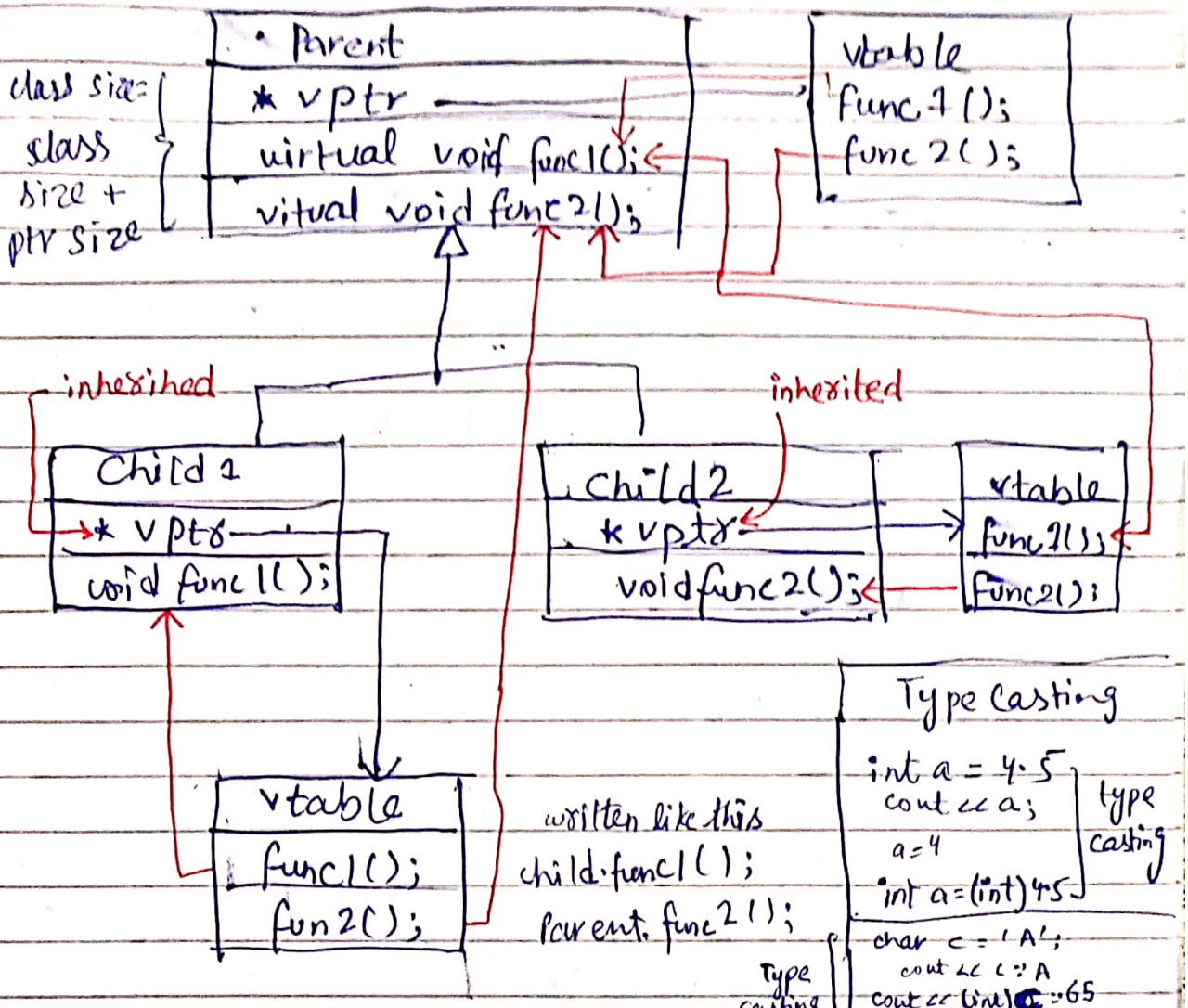


Virtual tables (vtable) and virtual pointer (vptr)



Next page

virtual table (vtable) and virtual pointer (vptr)



Type Casting

```
int a = 4.5
cout << a;
a=4
int a=(int)4.5
```

type casting

Type casting concept

```
char c = 'A';
cout << c;  :: A
cout << (int)c;  :: 65
int a=(int)c or int a=c;
cout << a;  :: 65
cout << (char)a;  :: A
```

Polymorphism..... Up-Casting v/s Down Casting

Let:

Base class has two functions print() and getData() Derived class has one function display();

```
void main() {
    Base* obj = new Derived;
    obj->print(); ✓
    obj->display(); ✗
    Derived d1;
    Derived* d2 = (Derived*) &obj;
    d2->print(); ✓
    d2->display(); ✓
```

Type casting

```
Base* obj = (Base*) new Derived;
```

upcasting (child to parent conversion)

parent me func nhi he child me he to error aayega.

Down casting (Parent to child conversion)

OOP

24-5-22 Lec 25

Template Function(s)

```
int square(int number)
{
    return number * number;   // multiplication
}
double square(double number)
{
    return number * number;   // multiplication
}

void main()
{
    int i=10;
    double d = 5.6;
    cout << square(i) << endl;
    cout << square(d) << endl;
}
```

different parameter and return type

Merge krne ke liye template function likhte hain

→ **Template function(s):**
Template function(s) are generic function(s) that can work with any data type.

Syntax:

Header file: (user define)

```
template <class anyname>
```

let's convert previous overloaded function

into template function.

```
template < class T>
T square (T number)
{
   return number* number;
```

Scenario :: array passing

```
template <class S>

int count (S arr[ ], int s)
{
   int c = 0;
  for (int i= 0 ; i<s ; i++){
        if(arr [i] != 0) {
            c++;
          }
      }
    return c;
}

void main()
{
   int arr[5] = {1,2,3,4,5};
      cout << count (arr,5);
}
```

Scenario:- Passing two parameters (2nd parameter ki datatype = return type)

```
template <class S, class T>
T count (S a, T b)
```

OOP

26-5-22 Lec 26

Template Classes

let's discuss a scenario and then we will convert this into template class

```
template <class M>
class lec26
{ M
  int * list;        // if any one datamember of class is of template type whole class (member function) also become template.
  int maxsize;
public:
    lec26();
    lec26(int n);
    int^M getSpecificValue(int index);
    ~lec26();
  {
    cout<<"Destructor\n";
    // code for deleting dynamic memory
  }
};  // end of class
```

template <class M> → Syntax to write template class function outside the scope of class.

→ Sometimes template functionality will end after statement terminator of class because of its header file definition. In order to prevent error we again define <M>

```
lec 26<M>::lec 26()
{
    list = NULL;
    maxsize = 0;
}
template<class M>
<M>
lec26<M>::lec 26(int n)
{
    list = new int[n];
    maxsize = n;
}
template<class M>
M
int lec 26<M>::getSpecificValue(int index)
{
    // assuming array is fill and index is valid
    return list[index];
}
void main(){ lec 26<int> obj1(10);
    lec26 obj1(10);
    cout << obj1.getSpecificValue();
}
```

Now let's convert previous class to template class that can deal with any type of array.

→ Exception in program / code

★ Throwing exception

let's discuss an example:

```
double division (int numerator, int denominator)
{
    if (denominator == 0)
    {
        cout<<"denominator can not be zero\n";
        return 0; -> dummy value return
                     compiler-fool
    }
  else
    {
    return static_cast<double>(numerator/denominator);
    }
} // end of function
```

problem:

0 can be a valid value.

Solution:-

Throw exception

→ Rewriting previous function with exception error

```
double division(int numerator, int denominator)
    {
throw point  if (denominator == 0) {
                 throw "denominator can not be zero\n";
             }
    else
```

throw → keyword

```
{
    // same as previous
}
```

OOP

31-5-22 Lec 27

Exception Handling

Ch#16
Starting out with C++
Tony Gaddies
8th edition

→ Exception handling:

Syntax:

```
try
{
    code that throws
    the exception.
}
catch(exception parameter)
{
    code to handle
    exception
}
```

```
double divide(int numerator, int denominator)
{
  if (denominator == 0)
    {
      string s = "Denominator can not be zero!";
      throw s;
    }
  else
    {
      return static_cast<double>(numerator/denomin
    }
} // end of function
```

```
void main()
{
   try
    {
      divide(4,0);
    }
  catch (string e)
    {
      cout << e << endl;
    }
}
```

→ Exception handling of classes

```
class Sample
{
   int a;
public:
   class NegativeValue
   {
   };
   void setA(int val)
   {
     if (val < 0)
     {
       throw NegativeValue();
     } // end of if
     else
     {
       a = val;
     }
```

(A) we make a dummy class

we are creating instance of dummy class

(B) → exception point

```
    } //end of setA
int getA()
  {
    return a;
  }
}; //end of sample class

void main()
  {
    Sample s;
    try
      {
        s.setA(-10);
      }
    catch(Sample::NegativeValue)
    {
      cout<<"value is negative\n";
    }
  } //end of main
```

(c)

→ if we want to throw the value at which the exception generated, do the change at following locations.

At (A)

```
class NegativeValue
{
  int e;
public:
    NegativeValue(int v)
  {
    e=v;
  }
```

```
int getvalue()
{
  return e;
}
```

At (B)

```
throw NegativeValue(val);
```
→ parameterized constructor

At (C)

```
catch(Sample::NegativeValue obj)
{
  cout<<obj.getValue()<<endl;
}
```
→ output: -10

OOP

2-6-22 Lecture 28

Standard Template Library-STL

STL is offered by C++ that provides multiple algorithms - functions and data structures - classes.

→ **Container**: Datatype that STL offers.

Type of Containers

1- Sequence container :-

→ Store data in specific order or sequence
e.g arrays, vector, deques, list etc

2- Associative Container :-

→ can store data of customs data types
e.g set, multiset, map, multimap

Extra:

Vectors in C++

Vector does not have size limits, the size is continous

```
#include<vector>
void main(){
vector<int> abc;      -> no initial size
vector<int>abc(10);   -> 10 values, starting size
vector<int> abc(10,2);
```

initial size; Initial value at all points where 2 are stored.

Ch#7 Tony gadis

max limit = when storage is full